SCHETT MATTHIAS

SEN-ÜBUNG 2.7

Inhaltsverzeichnis

Aufgabe 1

Lösungsidee

Es soll die Wetter Stations Verwaltung aus Übung 4 mithilfe der std::list neu implmentiert werden. Daher wird die Wetterstations Klasse unverändert übernommen. In der WeatherStations[1] Klasse werden jedoch einige Veränderungen durchgeführt. Es gibt nun zum Beispiel nur mehr eine Membervariable, mWeatherStations vom Type std::list<WeatherStation>. Durch die Verwedung von std::list[2] ergeben sich auch für die Implementierung einige Unterschiede. Weiters gilt es anzumerken, dass durch die Verwendung von std::list der Default Konstruktor, Default Copy Konstruktor, Default Destruktor und Default Assignement Operator verwendet werden kann, da Beispielsweise die Konstruktoren, die jeweiligen Konstruktoren der Membervariablen aufrufen.

[1] Zuständig für die Verwaltung von Wetterstationen

[2] std::list ist die Standardimplementierung einer doppelt verketteten List

Listing 1: Alte Implementierung Add

```
1  bool WeatherStations::Add(WeatherStation const &ws){
2      if(mNumberOfStations < mMaxNumber){
3          mStationArray[mNumberOfStations] = ws;
4          ++mNumberOfStations;
5          return true;
6      }
7      return false;
8  }
```

Listing 2: Neue Implementierung Add

```
1  void WeatherStations::Add(WeatherStation const &ws){
2      if(mWeatherStations.empty()){
3          mWeatherStations.push_front(ws);
4      } else{
5          // using standard functions just push it to the back then sort and
               remove duplicates
6          mWeatherStations.remove_if(RemoveWithName(ws));
7          mWeatherStations.push_back(ws);
8          mWeatherStations.sort(sortByName);
9      }
10 }
```

Auch die 2 Hilfsmethoden searchForColdest und searchForWarmest wurden neu implementiert

Listing 3: Implementierung der searchForColdest-Methode

```
1  WeatherStation WeatherStations::searchForColdest() const{
2      if(!mWeatherStations.empty()){
3          std::list<WeatherStation>::const_iterator it = std::min_element(
               mWeatherStations.begin(), mWeatherStations.end(), findColdest);
4          return *it;
5      } else {
6          throw WeatherException("No stations are defined");
7      }
8  }
```

Zu guter Letzt musste auch die PrintAll-Methode angepasst werden:

Listing 4: Implementierung PrintAll

```
1  void WeatherStations::PrintAll(std::ostream &out) const{
2      if(!mWeatherStations.empty()){
3          for (std::list<WeatherStation>::const_iterator it = mWeatherStations.
               begin(); it != mWeatherStations.end(); ++it) {
4              it->Print(out);
5          }
6      } else{
7          out << "The list is empty" << std::endl;
8      }
9  }
```

Der Code findet sich ab Anhang Aufgabe 1.[3]

[3] Der Code der WeatherStation Klasse wird da diese nicht verändert wurde nicht mit angehängt.

Testfälle

Der Testreiber testet die Funktionnalität indem er zuerst ein Objekt erstellt, vier Wetterstationen versucht hinzuzufügen und anschließend sämtliche Methoden aufruft.

Ausgabe

Listing 5: Testfall Ausgabe

```
1  Test Add
2
3  Test PrintAll
4  Name: WS1 Temperatur: 18.3 Luftfeuchtigkeit: 99
5  Name: WS2 Temperatur: 6.3 Luftfeuchtigkeit: 12
6  Name: WS3 Temperatur: -17 Luftfeuchtigkeit: 14
7
8  Test Getter
9  Should return 3: 3
10
11 Test Duplicates
12 Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
13
```

```
14 Add and Print All
15 Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
16 Name: WS2 Temperatur: 6.3 Luftfeuchtigkeit: 12
17 Name: WS3 Temperatur: -17 Luftfeuchtigkeit: 14
18
19 Test copy constr
20
21 Coldest of original: Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
22
23 Coldest of copy: Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
24
25 Test Assignment operator
26
27 Warmest of original: Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
28
29 Warmest of copy: Name: WS1 Temperatur: -19 Luftfeuchtigkeit: 22
```

Aufgabe 2

Lösungsidee

Es sollen verschiedene STL Algorithmen getestet werden. Dazu wird ein std::vector angeleget, der die in der Aufgabenstellung beschriebene Klasse Element aufnimmt. Dieser std::vector wird mittels des Zufallszahlengenerators befüllt. Anschließend wird mittles des Copy Konstruktors eine Kopie erstellt und mittels des std::copy Algorithmus und eines std::back_inserters eine std::list als Kopie erstellt. Diese 3 Container werden unterschiedlich sortiert, das Original mit std::sort, der Kopierte std::vector mit std::stable_sort und die Liste mit std::list.sort.

Über das Resultat lässt sich sagen, dass std::stable_sort und std::list.sort beide das gleiche Ergebnis liefern, da beide eine Liste die nach den Zufalsswerten sortiert ist und gleichzeitig nach dem Index liefern. Die Begürndung für dieses Verhalten findet sich in der offizielen Dokumentation[4].

Der letzte Test erfolgt mit equal_range der einen Bereich mit den gleichen Werte zurückgibt, aus welchem sich dann errechnen lässt, wie oft eine bestimmte Zufallszahl vorkommt. Der Code findet sich ab Anhang Aufgabe 1.

[4] Sorts the elements in the range (first,last) into ascending order, like sort, but stable_sort preserves the relative order of the elements with equivalent values.(Quelle: `http://www.cplusplus.com/reference/algorithm/stable_sort/` Datum: 18-05-2013)

Testfälle

Listing 6: Testfall Ausgabe

```
1
2 Test with filled vector
3 Unsorted values:
4 5-0  5-1  3-2  1-3  6-4  2-5  6-6  8-7  2-8  3-9  4-10  7-11  3-12  4-13  8-14
      5-15  2-16  4-17  1-18  8-19  0-20  5-21  7-22  4-23  7-24  4-25  7-26
    1-27  9-28  9-29  1-30  9-31  8-32  9-33  4-34  8-35  9-36  2-37  7-38
    6-39
5
6 Ascending sorted values:
7 0-20  1-18  1-3  1-30  1-27  2-37  2-5  2-8  2-16  3-2  3-9  3-12  4-34  4-10
     4-13  4-25  4-23  4-17  5-1  5-15  5-0  5-21  6-6  6-4  6-39  7-22  7-24
     7-26  7-11  7-38  8-19  8-14  8-7  8-32  8-35  9-28  9-29  9-31  9-33
    9-36
8
```

```
Ascending stable_sorted values:
0-20  1-3  1-18  1-27  1-30  2-5  2-8  2-16  2-37  3-2  3-9  3-12  4-10  4-13
    4-17  4-23  4-25  4-34  5-0  5-1  5-15  5-21  6-4  6-6  6-39  7-11  7-22
    7-24  7-26  7-38  8-7  8-14  8-19  8-32  8-35  9-28  9-29  9-31  9-33
   9-36

Ascending list sorted values:
0-20  1-3  1-18  1-27  1-30  2-5  2-8  2-16  2-37  3-2  3-9  3-12  4-10  4-13
    4-17  4-23  4-25  4-34  5-0  5-1  5-15  5-21  6-4  6-6  6-39  7-11  7-22
    7-24  7-26  7-38  8-7  8-14  8-19  8-32  8-35  9-28  9-29  9-31  9-33
   9-36

Frequency of random numbers
random number(0): 1
random number(1): 4
random number(2): 4
random number(3): 3
random number(4): 6
random number(5): 4
random number(6): 3
random number(7): 5
random number(8): 5
random number(9): 5
random number(10): 0

Test with empty vector
Unsorted values:


Ascending sorted values:


Ascending stable_sorted values:


Ascending list sorted values:


Frequency of random numbers
random number(0): 0
random number(1): 0
random number(2): 0
random number(3): 0
random number(4): 0
random number(5): 0
random number(6): 0
random number(7): 0
random number(8): 0
random number(9): 0
```

```
52 random number(10): 0
```

Anhang A
Aufgabe 1

Listing A.1: Header der Wetterstationsverwaltung

```
1  ////////////////////////////////////////
2  // Workfile    : WeatherStations.h
3  // Author      : Matthias Schett
4  // Date        : 12-04-2013
5  // Description : WeatherStation Manager class
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 #include "WeatherStation.h"
11 #include <iostream>
12 #include <exception>
13 #include <list>
14
15 class WeatherException : public std::exception
16 {
17 public:
18     WeatherException(const char* errMessage):errMessage_(errMessage){}
19     // overriden what() method from exception class
20     const char* what() const throw() { return errMessage_; }
21
22 private:
23     const char* errMessage_;
24 };
25
26 class WeatherStations {
27
28 private:
29
30     // member variables
31     std::list<WeatherStation> mWeatherStations;
32
33     // private helper methods
34     // Search for the coldes station
35     WeatherStation searchForColdest() const;
```

```
36      // Search for the warnest station
37      WeatherStation searchForWarmest() const;
38
39  public:
40
41      //*********************************
42      // Method:     Add
43      // FullName:   WeatherStations::Add
44      // Access:     public
45      // Returns:    bool
46      // Qualifier:
47      // Parameter: WeatherStation const & ws
48      // Adds a weather station
49      //*********************************
50      void Add(WeatherStation const &ws);
51
52      //*********************************
53      // Method:     Remove
54      // FullName:   WeatherStations::Remove
55      // Access:     public
56      // Returns:    bool
57      // Qualifier:
58      // Parameter: WeatherStation const & ws
59      // Removes a weather station
60      //*********************************
61      bool Remove(WeatherStation const& ws);
62
63      // Returns the number of weather stations
64      size_t GetNrStations() const;
65
66
67      //*********************************
68      // Method:     PrintAll
69      // FullName:   WeatherStations::PrintAll
70      // Access:     public
71      // Returns:    void
72      // Qualifier: const
73      // Parameter: std::ostream & out
74      // Prints all weather stations
75      //*********************************
76      void PrintAll(std::ostream &out) const;
77
78
79      //*********************************
80      // Method:     PrintColdest
81      // FullName:   WeatherStations::PrintColdest
82      // Access:     public
83      // Returns:    void
84      // Qualifier: const
```

```
85      // Parameter: std::ostream & out
86      // Prints the coldest weather station
87      //**********************************
88      void PrintColdest(std::ostream &out) const;
89
90      //**********************************
91      // Method:    PrintWarmest
92      // FullName:  WeatherStations::PrintWarmest
93      // Access:    public
94      // Returns:   void
95      // Qualifier: const
96      // Parameter: std::ostream & out
97      // Prints the warmest weather station
98      //**********************************
99      void PrintWarmest(std::ostream &out) const;
100
101 };
```

Listing A.2: Implementierung der Wetterstationsverwaltung

```
1  ////////////////////////////////////////
2  // Workfile    : WeatherStations.cpp
3  // Author      : Matthias Schett
4  // Date        : 12-04-2013
5  // Description : WeatherStation Manager class
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9  #include "WeatherStations.h"
10 #include <algorithm>
11
12 bool findColdest(WeatherStation const &ws1, WeatherStation const &ws2){
13     return ws1.GetCelsius() < ws2.GetCelsius();
14 }
15
16 bool findWarmest(WeatherStation const &ws1, WeatherStation const &ws2){
17     return ws1.GetCelsius() > ws2.GetCelsius();
18 }
19
20 bool sortByName(WeatherStation const &ws1, WeatherStation const &ws2){
21     return ws1.GetName() < ws2.GetName();
22 }
23
24 class RemoveWithName{
25 private:
26
27     WeatherStation wsIntern;
28
29 public:
30     RemoveWithName(WeatherStation const & ws) : wsIntern(ws){
```

```
31      }
32
33      bool operator () (WeatherStation const &ws1){
34          return ws1.GetName() == wsIntern.GetName();
35      }
36
37  };
38
39  void WeatherStations::Add(WeatherStation const &ws){
40      if(mWeatherStations.empty()){
41          mWeatherStations.push_front(ws);
42      } else{
43          // using standard functions just push it to the back then sort and
                remove duplicates
44          mWeatherStations.remove_if(RemoveWithName(ws));
45          mWeatherStations.push_back(ws);
46          mWeatherStations.sort(sortByName);
47      }
48  }
49
50  // Returns the number of weather stations
51  size_t WeatherStations::GetNrStations() const{
52      return mWeatherStations.size();
53  }
54
55  // Prints all weather stations
56  void WeatherStations::PrintAll(std::ostream &out) const{
57      if(!mWeatherStations.empty()){
58          for (std::list<WeatherStation>::const_iterator it = mWeatherStations.
                begin(); it != mWeatherStations.end(); ++it) {
59              it->Print(out);
60          }
61      } else{
62          out << "The list is empty" << std::endl;
63      }
64  }
65
66  // Prints coldest/warmest weather station
67  void WeatherStations::PrintColdest(std::ostream &out) const {
68      try{
69          searchForColdest().Print(out);
70      } catch(WeatherException& e){
71          std::cout << e.what() << std::endl;
72      }
73  }
74  void WeatherStations::PrintWarmest(std::ostream &out) const{
75      try{
76          searchForWarmest().Print(out);
77      } catch(WeatherException& e){
```

```
78          std::cout << e.what() << std::endl;
79      }
80  }
81
82  WeatherStation WeatherStations::searchForColdest() const{
83      if(!mWeatherStations.empty()){
84          std::list<WeatherStation>::const_iterator it = std::min_element(
                mWeatherStations.begin(), mWeatherStations.end(), findColdest);
85          return *it;
86      } else {
87          throw WeatherException("No stations are defined");
88      }
89  }
90
91  WeatherStation WeatherStations::searchForWarmest() const{
92      if(!mWeatherStations.empty()){
93          std::list<WeatherStation>::const_iterator it = std::max_element(
                mWeatherStations.begin(), mWeatherStations.end(), findWarmest);
94          return *it;
95      } else {
96          throw WeatherException("No stations are defined");
97      }
98  }
```

Listing A.3: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 12-04-2013
5  // Description : WeatherStation Manager class
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <vld.h>
11 #include <iostream>
12 #include "WeatherStations.h"
13
14 using namespace std;
15
16 int main(){
17
18     WeatherStation ws1 ("WS1", 18.3, 99);
19     WeatherStation ws2 ("WS2", 6.3, 12);
20     WeatherStation ws3 ("WS3", -17, 14);
21     WeatherStation ws4 ("WS1", -19, 22);
22
23     WeatherStations wsManager;
24
```

```
25      cout << "Test Add" << endl;
26
27      wsManager.Add(ws1);
28      wsManager.Add(ws2);
29      wsManager.Add(ws3);
30      wsManager.Add(ws1);
31
32      cout << endl;
33      cout << "Test PrintAll" << endl;
34
35      wsManager.PrintAll(cout);
36      cout << endl;
37      cout << "Test Getter" << endl;
38
39      cout << "Should return 3: " << wsManager.GetNrStations() << endl;
40      cout << endl;
41
42      cout << "Test Duplicates" << endl;
43      cout << ws4 << endl;
44      cout << "Add and Print All" << endl;
45      wsManager.Add(ws4);
46      wsManager.PrintAll(cout);
47
48      cout << endl;
49      cout << "Test copy constr" << endl;
50      cout << endl;
51      WeatherStations wsManager1 (wsManager);
52
53      cout << "Coldest of original: ";
54      wsManager.PrintColdest(cout);
55
56      cout << endl << "Coldest of copy: ";
57      wsManager1.PrintColdest(cout);
58      cout << endl;
59      cout << endl << "Test Assignment operator" << endl;
60
61      WeatherStations wsManager2 = wsManager;
62      cout << endl;
63      cout << "Warmest of original: ";
64      wsManager.PrintWarmest(cout);
65
66      cout << endl << "Warmest of copy: ";
67      wsManager2.PrintWarmest(cout);
68
69      cin.get();
70      return 0;
71  }
```

Anhang B
Aufgabe 2

Listing B.1: Header der Elementklasse

```
1  ////////////////////////////////////
2  // Workfile    : Element.h
3  // Author      : Matthias Schett
4  // Date        : 16-05-2013
5  // Description : Stl algorithm tests
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include "RandomGen.h"
11 #include <ostream>
12
13 class Element{
14
15 private:
16     int mRandNum;
17     int mIndex;
18
19 public:
20
21     Element(int randNum, int index);
22
23     // Access Methods
24     int getRandNum() const;
25     int getIndex() const;
26
27     bool operator<(Element const &e2);
28 };
29
30 inline std::ostream &operator << (std::ostream &out, const Element & elem){
31     out << elem.getRandNum() << "-" << elem.getIndex() << "  ";
32     return out;
33 }
34
35 inline bool operator<(Element const &e1, Element const &e2){
```

```
36      return ( e1.getRandNum() < e2.getRandNum() );
37  }
```

Listing B.2: Implementierung der ElementKlasse

```
1   ////////////////////////////////////
2   // Workfile     : Element.cpp
3   // Author       : Matthias Schett
4   // Date         : 16-05-2013
5   // Description  : Stl algorithm tests
6   // Remarks      : -
7   // Revision     : 0
8   ////////////////////////////////////
9
10  #include "Element.h"
11
12  using namespace std;
13
14
15  Element::Element(int randNum, int index) : mRandNum(randNum), mIndex(index){
16  }
17
18
19  int Element::getRandNum() const{
20      return mRandNum;
21  }
22
23  int Element::getIndex() const{
24      return mIndex;
25  }
26
27  bool Element::operator<(Element const &e2){
28      return ( getRandNum() < e2.getRandNum() ) ;
29  }
```

Listing B.3: Header des STL Tests

```
1   ////////////////////////////////////
2   // Workfile     : StlTest.h
3   // Author       : Matthias Schett
4   // Date         : 16-05-2013
5   // Description  : Stl algorithm tests
6   // Remarks      : -
7   // Revision     : 0
8   ////////////////////////////////////
9
10  #include "RandomGen.h"
11  #include "Element.h"
12  #include <vector>
13  #include <ostream>
14  #include <list>
```

```
15
16 int const minVal = 0;
17 int const maxVal = 10;
18
19 //**********************************
20 // Method:    startStlTest
21 // FullName:  startStlTest
22 // Access:    public
23 // Returns:   void
24 // Qualifier:
25 // Starts the stl test specified in the task
26 //**********************************
27 void startStlTestOK();
28
29 //**********************************
30 // Method:    startStlTest
31 // FullName:  startStlTest
32 // Access:    public
33 // Returns:   void
34 // Qualifier:
35 // Starts the stl test specified in the task with empty vectors
36 //**********************************
37 void startStlTestNOK();
38
39 //**********************************
40 // Method:    printVector
41 // FullName:  printVector
42 // Access:    public
43 // Returns:   void
44 // Qualifier:
45 // Parameter: ostream & os
46 // Parameter: vector<Element>
47 // Prints the given vector
48 //**********************************
49 void printVector(std::ostream &os, std::vector<Element> elemVec);
50
51
52 //**********************************
53 // Method:    printList
54 // FullName:  printList
55 // Access:    public
56 // Returns:   void
57 // Qualifier:
58 // Parameter: std::ostream & os
59 // Parameter: std::list<Element> elemVec
60 // Prints the given list
61 //**********************************
62 void printList(std::ostream &os, std::list<Element> elemVec);
63
```

```
64 //***********************************
65 // Method:      countFrequencyOfNumbers
66 // FullName:    countFrequencyOfNumbers
67 // Access:      public
68 // Returns:     void
69 // Qualifier:
70 // Parameter: std::vector<Element> elemVec
71 // Counts how often a specific number occurs inside the container
72 //***********************************
73 void countFrequencyOfNumbers(std::vector<Element> elemVec);
74
75 //***********************************
76 // Method:      createRandomNumbers
77 // FullName:    createRandomNumbers
78 // Access:      public
79 // Returns:     std::vector<Element>
80 // Qualifier:
81 // Parameter: int numOfRandomNum
82 // Creates a vector with filled with random numbers
83 //***********************************
84 std::vector<Element> createRandomNumbers(int numOfRandomNum);
```

Listing B.4: Implementierung des STL Tests

```
1  ////////////////////////////////////////
2  // Workfile    : StlTest.cpp
3  // Author      : Matthias Schett
4  // Date        : 16-05-2013
5  // Description : Stl algorithm tests
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////
9
10 #include "StlTest.h"
11 #include <algorithm>
12 #include <iterator>
13 #include <iostream>
14
15 using namespace std;
16
17 struct sortCriteria{
18     bool operator() (const Element &a, const Element &b){
19         return ( a.getRandNum() < b.getRandNum() ) ;
20     }
21 };
22
23 struct equalCriteria{
24     bool operator() (const Element &a, const Element &b){
25         return ( a.getRandNum() == b.getRandNum() ) ;
26     }
```

```
27 };
28
29 void printVector(std::ostream &os, std::vector<Element> elemVec){
30     std::ostream_iterator<Element> out_it (os);
31     std::copy ( elemVec.begin(), elemVec.end(), out_it );
32
33 }
34
35 void printList(std::ostream &os, std::list<Element> elemList){
36     std::ostream_iterator<Element> out_it (os);
37     std::copy ( elemList.begin(), elemList.end(), out_it );
38 }
39
40 std::vector<Element> createRandomNumbers(int numOfRandomNum){
41     vector<Element> elemVec;
42
43     rgen::Init();
44
45     for(int i = 0; i < numOfRandomNum; i++){
46         elemVec.push_back(Element(rgen::GetRandVal(minVal, maxVal), i));
47     }
48
49     return elemVec;
50 }
51
52 void countFrequencyOfNumbers(std::vector<Element> elemVec){
53
54     std::pair<std::vector<Element>::iterator,std::vector<Element>::iterator>
        bounds;
55     int freq[11] = {0};
56
57     for(int i = 0; i <= 10; i++){
58         bounds = std::equal_range(elemVec.begin(), elemVec.end(), Element(i,0)
            );
59         freq[i] = (bounds.second - elemVec.begin()) - (bounds.first - elemVec.
            begin());
60     }
61
62     for(int i = 0; i < 11; i++){
63         cout << "random number(" << i << "): " << freq[i] << endl;
64     }
65 }
66
67 void doTask(vector<Element> baseVec){
68
69     cout << "Unsorted values:" << endl;
70     printVector(std::cout, baseVec);
71
72     vector<Element> elemVec2(baseVec);
```

```
73
74      list<Element> elemList;
75      std::copy(baseVec.begin(), baseVec.end(), back_inserter(elemList));
76
77      std::sort(baseVec.begin(), baseVec.end(), sortCriteria());
78      std::stable_sort(elemVec2.begin(), elemVec2.end(), sortCriteria());
79      elemList.sort(sortCriteria());
80
81      cout << endl << endl;
82      cout << "Ascending sorted values:" << endl;
83      printVector(cout, baseVec);
84
85      cout << endl << endl;
86      cout << "Ascending stable_sorted values:" << endl;
87      printVector(cout, elemVec2);
88
89      cout << endl << endl;
90      cout << "Ascending list sorted values:" << endl;
91      printList(cout, elemList);
92
93      cout << endl << endl;
94      cout << "Frequency of random numbers" << endl;
95      countFrequencyOfNumbers(baseVec);
96  }
97
98  void startStlTestOK(){
99      cout << endl << "Test with filled vector" << endl;
100     vector<Element> baseVec = createRandomNumbers(40);
101     doTask(baseVec);
102 }
103
104 void startStlTestNOK(){
105     cout << endl << "Test with empty vector" << endl;
106     vector<Element> baseVec;
107     doTask(baseVec);
108 }
```

Listing B.5: Testtreiber

```
1  ////////////////////////////////////
2  // Workfile    : Main.cpp
3  // Author      : Matthias Schett
4  // Date        : 16-05-2013
5  // Description : Stl algorithm tests
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////
9
10 #include <iostream>
11 #include "StlTest.h"
```

```
12
13 using namespace std;
14
15 int main(){
16
17     startStlTestOK();
18
19     startStlTestNOK();
20
21     cin.get();
22     return 0;
23 }
```